Comparing Model-based and Data-driven controllers for an autonomous vehicle task

Erwin Jose Lopez Pulgarin[1](0000-0001-9927-6688), Tugrul Irmak[1], Joel Variath Paul[1], Arisara Meekul[1], Guido Herrmann[1](0000-0001-5390-4538), and Ute Leonards[2](0000-0001-6143-7466)

[1] Mechanical Engineering, University of Bristol, Bristol, UK
erwin.lopez@bristol.ac.uk, g.herrmann@bristol.ac.uk,
[2] Experimental Psychology, University of Bristol, Bristol, UK
ute.leonards@bristol.ac.uk

Abstract. The advent of autonomous vehicles comes with many questions from an ethical and technological point of view. The need for high performing controllers, which show transparency and predictability is crucial to generate trust in such systems. Popular data-driven, black box-like approaches such as deep learning and reinforcement learning are used more and more in robotics due to their ability to process large amounts of information, with outstanding performance, but raising concerns about their transparency and predictability. Model-based control approaches are still a reliable and predictable alternative, used extensively in industry but with restrictions of their own. Which of these approaches is preferable is difficult to assess as they are rarely directly compared with each other for the same task, especially for autonomous vehicles. Here we compare two popular approaches for control synthesis, model-based control i.e. Model Predictive Controller (MPC), and data-driven control i.e. Reinforcement Learning (RL) for a lane keeping task with speed limit for an autonomous vehicle; controllers were to take control after a human driver had departed lanes or gone above the speed limit. We report the differences between both control approaches from analysis, architecture, synthesis, tuning and deployment and compare performance, taking overall benefits and difficulties of each control approach into account.

1 Introduction

Current trends in automotive driver-assist systems indicate an increase in vehicle autonomy, making cars more like robots. Even when considering current robotics standards for personal robots like ISO 13482:2014 [8], semi-autonomous and autonomous vehicles fall into the categories of Person Carrier Robots.

Autonomous and semi-autonomous vehicles are areas of relevant commercial interest, with a multitude of ethical and technical challenges yet to be solved. One issue that remains unsolved is how to create controllers that integrate a large amount of heterogeneous sensor data (e.g. cameras, range sensors, vehicle sensors and driver sensors) and that are able to deal with highly complex and changing environments.

A large variety of control algorithms and approaches has been proposed to solve this problem in both real and simulated environments[19] [13] [21]. We can divide them into two categories, model based and data-driven. The former category belongs to a classical approach for industrial control; it uses a plant model and represents a mathematical description of the dynamics surrounding the system with a controller shaping those dynamics to meet a set of predefined requirements. The latter is based on an approach from computer science inspired by the human process of learning, subsequent decisions[7] and actions, i.e. the current artificial intelligence paradigm; available measurements or estimations are used to create a relation between the way we can manipulate the system (actions) and its measurements (states) by rewarding or punishing certain behaviours, with limited to no knowledge about the system itself.

In the domain of autonomous and semi-autonomous systems, some of the prominent algorithms used in each category are Model Predictive Control (MPC)[15] [10] for the model-based and Reinforcement Learning [3] [18] by actor-critic algorithms like Deep Deterministic Policy Gradients (DDPG) for the data-driven models.

Both approaches have their own benefits and hindrances during the possible stages of control design (i.e. analysis, architecture design, synthesis, tuning and deployment), requiring sets of skills and resources (e.g. computational time); yet to the best of our knowledge, they have never been directly compared for a set of identical tasks related to autonomous vehicles. Automatic control of the speed of a vehicle (i.e. cruise control) and maintaining a vehicle in the centre of the lane (i.e. lane keeping) are typical common tasks for autonomous vehicles and subsequent advanced driver assistance systems (ADAS). For automatic control systems, it is often the case that initialisation time is prohibitively long as in the range of seconds. For vehicle control and ADAS, it means that the controllers need to be run in parallel whilst the human is driving, and control be switched either manually or automatically. Switching between controllers is not a trivial task. The transients caused by controller switching can produce instabilities. The non-responsiveness of the vehicle to the ADAS inputs while the human driver is in control could cause actuator saturation. An abrupt handover of control to the ADAS could result in a violent response of the vehicle such as a spin, collision or loss of control. Speed has been linked to switching performance in autonomous vehicles scenarios [21].

In this paper, we applied separately MPC and DDPG to a cruise control and lane keeping task for a semi-autonomous vehicle or Advanced Driver Assist System (ADAS)[9]. Specifically, the task requires to switch control from a human driver to an automatic controller based on a performance metric. Here we present a one-on-one comparison between the two design methodologies during the different steps of the process and analyse the obtained performance; comments and ideas on how to improve each controller are also provided. Concepts around control design, the different control design approaches, algorithms and simulation models are explained in Section 2 and 3. Section 4 analyses and compares the results, and Section 5 talks about conclusions and future work.

2 Simulation Environment

The Open Car Racing Simulator [2] (TORCS) is an open source software. It allows to create its own artificial "drivers" that use virtual sensors to gather information about the road and the state of the vehicle; it also enables customized tracks and to connect to it through third party software (e.g. Python, MATLAB) with external pluggins, facilitating testing and prototyping.

The artificial driver controls steering wheel angle, throttle and brake commands. It implements a rich set of virtual measurements as sensor input. The sensors provide translational velocity in all 3 axis, 4 sensor readings of rotational wheel speed, number of engine cycles per minute, normalised position on track with regard to track width, angle of the long axis of the car relative to centre line of track and 19 sensor readings of track edge and car (i.e. laser radar).

Only the data-driven controller required the laser sensor input to work, whereas the model-based controller could work without it.

We used a variety of tracks with different difficulty levels (i.e. more aggressive geometry and thinner track width) to estimate the reliability of our two systems. Twelve different tracks were used in the experiment, with track widths ranging from 10 to 15 m, track length ranging from 2000 to 6000 m and curvatures ranging in average from 0.0016 to 0.02 m^{-1} with a maximum of 0.087 m^{-1}.

3 Control Design Approaches

Creating a control system is a complex process that involves several stages from the study and comprehension of the problem to the implementation of the obtained solution using a mathematical representation. The necessary steps will be taken as the important criteria for our comparison between the two different control approaches:

Analysis deals with understanding the system through looking at specific objectives (e.g. reference tracking, regulation control) and the requirements at hand. Both, system actions and system states (i.e. sensed or estimated system variables), respectively, are examined as well as the relationship between them.

Architecture Design has to do with selecting the structure of the proposed algorithm that meets the desired requirements. This step might involve the control law that will be implemented by the controller, and any or all initial parameter values for the control architecture (e.g. weights, starting point, look ahead time).

Synthesis deals with creating the controller with the architecture selected, generating a functional model that can enforce the control objective and requirements, to test and validate its performance.

Tuning involves the evaluation of the synthesized controller and the optimization of its performance. It deals with improving the controller in individual or all its performance requirements by means of tuning optimizing any value that does not change the architecture of the controller (e.g. weights, factors).

Deployment deals with taking the designed controller into the real application and establish how it can be best used with both sensors, actuators and an automatic computer.

3.1 Model Predictive Controller

Model Predictive Control (MPC) is an advanced technique that deals with a short term optimal target problem by combining a prediction and a control strategy. It approximates future behaviours of the target plant, compares the predicted state of the plant with a set of predefined objectives and computes the optimal control inputs to achieve the objectives, whilst respecting the plant's constraints.

MPCs have been successful in semi-autonomous and autonomous vehicles, as they can handle Multi-Input-Multi-Output (MIMO) systems with controller input and state variable constraints while considering the non-linear dynamics of vehicles [17].

They use an explicit internal model which is often a simplification of the actual system/plant dynamics to predict the outputs at future time instances (prediction horizon). The dynamics of any given system in discrete time can be expressed in the state-space form as:

$$x_{k+1} = Ax_k + Bu_k \tag{1}$$

$$y_{k+1} = CAx_k + CBu_k + d_{k+1} \tag{2}$$

$\boldsymbol{x}$ is the state vector of system variables, $\boldsymbol{A}$ the system matrix, $\boldsymbol{y}$ the output vector of observed variables, $\boldsymbol{B}$ the input matrix, $\boldsymbol{u}$ the control input vector, $\boldsymbol{C}$ the output matrix, $\boldsymbol{k}$ the time step and $\boldsymbol{d}$ the disturbance matrix.

At each control interval, the prediction horizon is shifted towards the future and a new set of predictions is made. Further details can be seen in [16].

For situations where time-varying plant characteristics exist, two of the many approaches for MPC implementation are: Implicit MPC with a linearised model around an operating model and Adaptive MPC with a model being updated at run time [11].

Published MPC algorithms for autonomous vehicles are often adaptive and use dynamic vehicle models (LPV) with a non-linear tyre model. Still, the use of kinematic models (i.e. linearised models around an operating point) for lateral control has been proposed and studied[12], showing that kinematic models with discretization time of $200ms$ show similar performance to dynamic models with discretization time of $100ms$.

Bicycle Model The dynamics of a road vehicle are complex[12], involving 3 translational and 3 rotational degrees of freedom (DOF's), which are often inter-dependent due to the nature of the vehicle's suspension. The coordinate system used during our modelling of the vehicle followed the 'SAE International: Vehicle Dynamics Terminology J670E' [1]. The non-linear nature of the suspension system, the effects of aerodynamic drag and lift forces and the nature of contact condition between the tyres and the road surface add additional complexity. The vehicle dynamics for the purposes of this project can be de-coupled in the longitudinal and lateral directions to produce independent mathematical models of either kinematic or dynamic nature.

Lateral dynamics of the vehicle can be simplified by approximating them to a 2 DOF bicycle model representing the lateral and yaw motions. A few

assumptions are made for the application of this simplified model, like constant longitudinal velocity, small slip angles (i.e. tyres operate in the linear region), no aligning moments in the tyres, no road gradient or bank angles, no lateral or longitudinal load transfer and no rolling or pitching motions. These assumptions are only valid at low speeds as the tyres experience significant slip and load transfers when cornering at high speeds.

Modelling the front pair of wheels as a single unit assumes that the steering angles on the left and right sides of the vehicle are the same. Automobiles in general use Ackerman geometry to ensure that the steering angles of the left and right wheels are such that they arc around a common instantaneous rolling centre for the vehicle. During cornering, the tyre's velocity has a component normal to the tyre plane, in addition to the component in the direction of motion. This is because the tyre experiences deformation and slip in the lateral direction. The angle between the direction of heading and the direction of travel (or ratio of lateral and longitudinal components of velocity) is known as slip angle. The cornering stiffness is defined as the ratio of change in lateral force and the change in slip angle. In commercial vehicles, the cornering stiffness of the front wheels is greater than that of the rear wheels, as it is implemented in this model. Further details available in [4].

Longitudinal dynamics govern the forward motion of the vehicle, constrained in the XZ plane. The longitudinal motion of the chassis accounts for aerodynamic drag and variations in the road incline to obtain a force balance like:

$$F_{forward} = M\frac{d}{dt}V_x + F_{aerodynamic} + F_{rolling} + F_{gravitational} + F_{friction} \quad (3)$$

$F_{aerodynamic}$ is aerodynamic drag, $F_{rolling}$ rolling resistance, $F_{gravitational}$ gravitational forces at slopes and $F_{friction}$ are frictional forces due to other mechanical losses of energy in the system. Vehicle parameters include mass $M = 1150\ kg$, density of air $\rho = 1.225\ kg/m^3$, drag coefficient $C_D = 0.8$, equivalent frontal area $A_F = 1.92\ m^2$, rolling resistance coefficients $C_{r0}/C_{r1} = 10^{-2}/10^{-4}$, coefficient of frictional damping $C_v = 7$, cornering stiffness of front tyre $C_f = 27\ kN/rad$, cornering stiffness of rear tyre $C_r = 21\ kN/rad$, distance of front and rear tyres (from centre of gravity) $l_f = l_r = 1.9\ m$ and moment of inertia about z-axis $I_z = 960\ kgm^2$.

Control Structure An Implicit MPC for lateral and an Adaptive MPC for longitudinal motion were created to control the vehicle.

For **lateral control**, the expression of the lateral force balancing equations in terms of lateral error y_e and heading error ψ_e produce:

$$\begin{bmatrix} \dot{y}_e \\ \ddot{y}_e \\ \dot{\psi}_e \\ \ddot{\psi}_e \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{(C_f+C_r)}{MV_x} & 0 & -V_x - \frac{C_f l_f - C_r l_r}{MV_x} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{C_f l_f - C_r l_r}{I_z V_x} & 0 & -\frac{C_f l_f^2 + C_r l_r^2}{I_z V_x} \end{bmatrix} \begin{bmatrix} y_e \\ \dot{y}_e \\ \psi_e \\ \dot{\psi}_e \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{C_f}{M} \\ 0 \\ \frac{C_f l_f}{I_z} \end{bmatrix} \delta_f \quad (4)$$

The obtained expression is a function of the longitudinal velocity of the car V_x. An Implicit MPC with a linearised model and an Adaptive MPC were tested.

The Implicit MPC was linearised about an operating point of $10m/s$ with a set of bounds for the manipulated variables (MV). Both controllers performed almost identically. Adaptive MPC is computationally intensive and runs into numerical errors at high velocities ($> 90km/h$). As there is no significant performance advantage in switching to an adaptive controller, the implicit MPC is selected [11].

The used control parameters were: Prediction Horizon ($H_p = 30$), Control Horizon ($H_c = 5$) and Sampling time ($T_s = 20ms$). The used tuning parameters were: MV min/max $= -0.66/0.66rad$, MV Rate min/max $= -0.8/0.8rads/s$, MV Scale factor $= 8$, $Q_n = diag[1\ 0.5]$, $R_n = 2$.

The **longitudinal controller** regulates the throttle and brakes to achieve its desired forward velocity. Both Implicit MPC and Adaptive MPC were tested, with the former having limited accuracy obtained due to not accommodating the variance of throttle demand with road terrain. The Adaptive MPC implements a dynamic model of the forward dynamics incorporating gravitational, aerodynamic and rolling resistances. The Linear Parameter-Varying Model (LPV) used depends on Angle of incline (Slope) α, Pedal constant F_p, Gear ratio G_r, Throttle input u, Longitudinal velocity V_x and the other longitudinal parameters. It can be expressed as:

$$\left[\dot{V}_x\right] = \left[-\left(\frac{g(sin(\alpha)+C_{r0})}{Vx} + \frac{(\rho C_D A_F V_x + 2C_v)}{2M}\right)\right]\left[V_x\right] + \left[F_p G_r\right] u_f \quad (5)$$

To reduce cornering tendencies due to turning whilst maintaining a stable cruise velocity, a strategy to adapt the velocity output demand was implemented. Road curvature is used to reduce the velocity whilst turning to improve vehicle handling. The minimum value between the cruise velocity and a threshold is used as a final cruise velocity. Empirical testing revealed that a threshold of $0.3/abs(RoadCurvature)$ produced the best results.

3.2 Reinforcement Learning

Reinforcement Learning is based on the idea that reinforcement, if given in the form of a reward or punishment, will help to optimise a behavioural response. This process of behavioural adaptation (learning) aims to maximise the cumulative reward extracted from the environment. The agent is said to navigate a Markov decision process (MDP)[20], as defined by the five tuple $\langle \mathtt{S}, \mathtt{A}, \mathtt{R}, \mathtt{P}, \gamma \rangle$: $\mathtt{S}$ is the set of states, $\mathtt{A}$ is the set of actions, $\mathtt{R}$ is the expected reward given a state action pair, $\mathtt{R}^{\mathtt{a}}_{\mathtt{s}} = \mathbb{E}[\mathtt{r}_{\mathtt{t}} \mid \mathtt{S}_{\mathtt{t}} = \mathtt{s}, \mathtt{A}_{\mathtt{t}} = \mathtt{a}]$, $\mathtt{P}$ is the state transition probability matrix $\mathtt{P}^{\mathtt{a}}_{\mathtt{ss}'} = \mathbb{E}[\mathtt{S}_{\mathtt{t}+1} = \mathtt{s}' \mid \mathtt{S}_{\mathtt{t}} = \mathtt{s}, \mathtt{A}_{\mathtt{t}} = \mathtt{a}]$ and γ is the discount factor in domain [0,1].

The goal of the agent is to extract the most reward from the MDP by deciding on the best action, given the state it is in. The actions of an agent, given the state it is in, are determined by the agent's policy. MDPs are time invariant and depend only on the current state and not the state history. Solving an MDP means finding an optimal policy which yields the optimal value. Policy evaluation can be done by Monte-Carlo (MC) Backup and Temporal Difference (TD) Backup[20]. Policy control, the process of iteratively finding a better policy,

can be done by algorithms such as SARSA or Q-learning[22], which creates a Q-value Q that relates actions a and states s through a reward.

As the number of states and actions increases, the number of table entries required increases exponentially. Physical systems are controlled in a continuous state action space, creating the need for continuous function approximators.

For physical systems, the idea of actor-critic was introduced [23]. The actor is the policy which is updated every iteration. The critic is the Q value, which is updated every iteration. Initial implementations also had the critic being updated by minimising the mean square difference between the approximation and the computed evaluated values. The actor is a Gaussian distribution over actions, given state where the mean and standard deviation is parametrised by a linear sum of weighted features.

The policy parameters are updated to maximise the cumulative reward extracted from the MDP, where each state transition yields a reward. The magnitude of this reward is determined by the reward function. Therefore, the reward function implicitly describes the optimal behaviour. Poorly designed reward functions can lead to oscillations in policy and convergence to poor policies. Generally, reward functions which give continuous reward are better behaved than those that have a large delay. An example of this is chess. A move in mid game can lead to victory later, but it may not be easy to identify the utility of this singular move among the whole trajectory.

Control Structure An actor critic reinforcement learning (RL) technique was used to develop a continuous state-action controller. The method used was based on deep deterministic policy gradient (DDPG) algorithm [14], integrating an additional Proportional-Integral loop to improve tracking performance.

The action value (critic) function and the policy (actor) are both approximated by two neural networks (NN) with two hidden layers each and a linear rectifier (relu) activation function. For the first and second hidden layer, 300 and 600 nodes were selected respectively based on improvements done on the original implementation [14]. The algorithm uses all 29 sensor inputs. The critic input to the first layer is the state and to the second layer is the action. The output is the action value. The matrice's weights are initialized using Xavier[5] initialisation. Policy parameters are tuned online using stochastic gradient descent.

The vehicle was trained on the track with the widest variety of turns (ie. straight, cambered, inclined and sharp). This increases the generality of the trained algorithm to other tracks. The aim of the DDPG is to control a vehicle to safely travel at a user specified cruising velocity, V_t whilst reducing the lateral distance from the centreline of the track, $|y_e|$.

The reward function [14] $R(s) = V_l(\cos\theta - |\sin\theta| - |y_e|)$ was used. V_l is the component of the vehicle's longitudinal velocity in the direction of the track. θ is the yaw angle. Additionally, a negative reward was given if the vehicle left the track. Lastly, the simulation was reset if no progress was made after 500 simulation steps. Only 2 of every 3 controllers synthesized had good performance (i.e. algorithm converged).

The DDPG allows for the development of a robust model-free controller for a high speed racer. However, it does not yield well to constraining the cruising

speed to a user reference. To tackle this, the DDPG+PI controller was developed which uses both reinforcement learning and classical proportional-integral control to overcome the limitations of both methods. A proportional gain of 8 and an integral gain of 0.05 were empirically obtained for the PI component. This allows good levels of robust tracking of the velocity and track demand.

4 Comparative Evaluation

The obtained compliant controllers showed similar nominal performance, following different design methodologies and taking different amounts of effort and resources in each stage.

4.1 Design procedure comparison

For the model-based controller or MPC:

- **Analysis:** A great deal of effort must be put into a mathematical model of the car and its movement in a dynamic framework. This is a time-consuming task, requiring a great amount of specialized knowledge.
- **Architecture:** Changes are restricted to the type of MPC being implemented (e.g. MPC, optimal MPC) and its parameters, which are generally not many.
- **Synthesis:** Time and computational resources are linked to the complexity of the mathematical formulation. It deals with automatically generating a code that solves a mathematical equation inside the control loop. The code could be too large and the equation too slow or unfeasible to be solved; Nonetheless, this can be included and validated before synthesis and dealt with.
- **Tuning:** Parameters can be manually tuned and are intuitive to the problem being solved, as they increase or decrease the size of the equation to be solved.
- **Deployment:** It needs a solver for an equation to be run inside the control loop, which could be difficult or unfeasible to do in a final implementation. Our implementation did not suffer of these problems, but running two MPCs simultaneously was computationally intensive.

For the data-driven controller or DDPG+PI:

- **Analysis:** No great amount of previous knowledge or expert knowledge is required. Just a small amount of information about the situation itself and how the states could be related is necessary to formulate a reward function.
- **Architecture:** After choosing a control architecture, no further work is necessary, and the algorithm can be implemented. The architecture itself allows for integration of any input into the controller. Yet, the selection of control structure is closely intertwined with a successful synthesis process. Careful iterative tests are needed to augment the right input signals and add any necessary control structures (e.g. additional PID blocks).
- **Synthesis:** Takes a large amount of time and computational power, as training requires replay and repetition of the driving scenario.

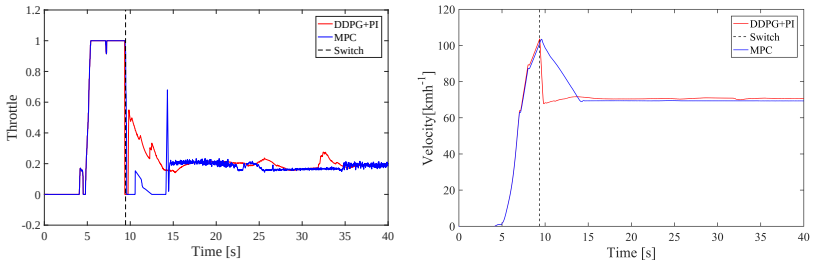
– **Tuning:** Takes a large amount of time, mainly due to the time it takes to synthesise each controller. Tuning of the parameters is not intuitive, and a rework of the reward function could be unpredictable in terms of design time and obtained performance.
– **Deployment:** It only needs to use the matrix weights, with sequential multiplication of the sensor inputs to create the control rule inside the control loop. Hence, the controller was easily portable between Python and MATLAB.

MPC does not have a native way of dealing with a great number of sensors, if the model does not incorporate them. RL can accommodate these sensors as part of the architecture. RL cannot accommodate big tracking errors or adapt to very different scenarios from the one the training was done, reason why adhoc methods like the PI controller improve the nominal performance.

Initial tests led to both controllers producing instabilities when control was given to them from human driving. As suggested in previous studies [21], a speed-dependent transition window (e.g. faster switching at higher speeds) with a 1^{st} order transfer function was implemented. Smooth transitions were generated, and instabilities were avoided.

4.2 Speed Limit Violation

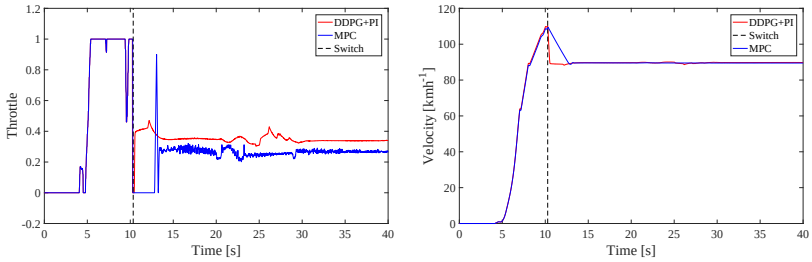
The vehicle is driven over a speed limit by a human driver past 2 different pre-defined maximum velocities ($100km/h$ and $110km/h$), with the controllers taking over and reducing the speed to $70km/h$ and $90km/h$ respectively.



(a) (b)

Fig. 1: Controllers takeover from driver due to over speeding at 100 kmh^{-1} (dotted black line). Throttle (a) and Vehicle velocity (b)

Both controllers manage to take the speed down successfully. Figures 1a and 2a show the change in throttle control signal when controllers take over. Figures 1b and 2b show the resulting change in velocity. Controllers taking-over happen around a time of $10s$. MPC shows a smoother deceleration in all cases. This is due to the action constraints applied on the rates of throttle and brake usage for MPC, compared to the possible integral term winds up as the over-speed value is reached for the RL controller.

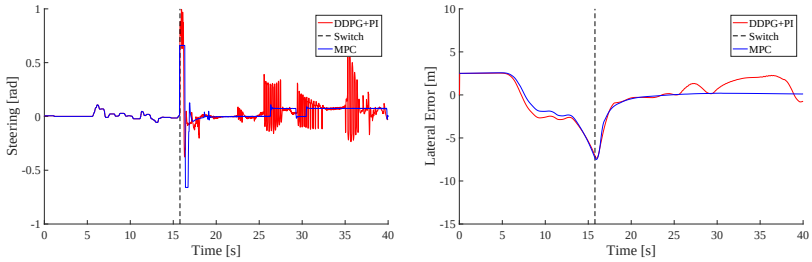


(a) (b)

Fig. 2: Controllers takeover from driver due to over speeding at 110 kmh^{-1} (dotted black line). Throttle (a) and Vehicle velocity (b)

4.3 Lane Departure

The vehicle is driven off track by a human driver at two different velocities ($60km/h$ and $110km/h$), with the controllers taking over once the lateral error increases over a threshold.

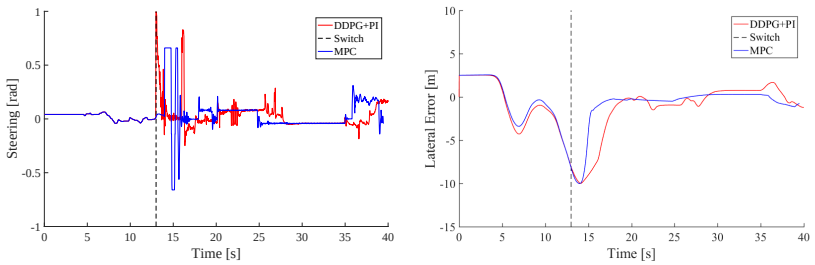


(a) (b)

Fig. 3: Controller takeover from driver due to lane departure at 60 kmh^{-1} (dotted black line). Steering Wheel Angle (a) and Vehicle lateral error (b)

Both controllers take the vehicle back on track. Figures 3a and 4a show the change in steering wheel control signal when controllers take over. Figures 3b and 4b show the resulting change in lateral error. Different performance traits between controllers and between nominal speeds were obtained, which complies to the intuitive idea that a car is more difficult to control at higher speeds. The RL controller shows higher oscillatory behaviour for the steering control, especially for the low speed case. The MPC controller shows consistently smooth steering control in both cases, but a more aggressive corrective manoeuvring for high speeds, which may lead to the vehicle spinning out of control.

The difference in behaviour could be linked to both controllers' architectures. The RL controller had a PI controller added on top to overcome its limitations



(a) (b)

Fig. 4: Controller takeover from driver due to lane departure at 110 kmh^{-1} (dotted black line). Steering Wheel Angle (a) and Vehicle lateral error (b)

with regards to error tracking, which allowed for better performance, especially at high speeds, but reduced performance for low speeds; as the RL controller was trained with higher speeds, tracking low speeds is specially challenging for it. The MPC controller allows to minimise the magnitude and frequencies of the lateral and longitudinal acceleration, resulting in smoother rides at all speeds, but reduced performance at high speeds due to the limitations of its internal model.

5 Conclusions

The design process for a controller used in semi-autonomous vehicles was presented, using two different design approaches (one model-based, the other data-driven). The controllers were directly compared with each other based on their performance when used to correct a speed limit violation or lane departure situation created by a human driver.

Both approaches were successful in producing a controller that keeps the vehicle in the lane and under a certain speed limit. Each had different benefits due to the design process, mainly with the MPC controller having direct ways to define control requirements and constraints, intuitive tuning and reduced synthesis time, and with the RL needing no expert knowledge about the model, support of large number of sensors and the high speed that its final implementation runs on.

For Lane departure correction, the RL controller seemed to perform better than the MPC controller as it changed vehicle position more gradually, resulting in a less aggressive return to the centreline. For the speed correction, however, the MPC controller seemed to perform better due to its control constraints, allowing a gentle deceleration.

The speed-dependent window smoothed the transition between manual human control and automatic controller; yet the controllers could generate discomfort to a human driver which should be considered when designing the controller. Restricting the vehicle's acceleration in the xy plane would reduce the possibility of inducing motion sickness or discomfort[6].

The fact that the model-based controller showed similar performance to the data-driven controller using less sensors is worth mentioning. This could be related to the predictive nature of the MPC, which achieves an estimation of the state of the vehicle further ahead in the track without the laser sensor; this estimation would not consider sudden changes, evidenced of that is the reduced performance of the controller when cornering at high speeds.

References

1. Automotive, S.: Vehicle Dynamics Terminology : J670E (1976)
2. Bernhard Wymann, Rémi Coulom, Christos Dimitrakakis, Eric Espié, Andrew Sumner: TORCS, The Open Racing Car Simulator (2014), `http://www.torcs.org`
3. Chen, C., Seff, A., Kornhauser, A., Xiao, J.: Deepdriving: Learning affordance for direct perception in autonomous driving. In: ICCV. pp. 2722–2730. IEEE (2015)
4. Gillespie, T.D.: Vehicle Dynamics. Warren dale (1997)
5. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: AISTATS. pp. 249–256 (2010)
6. Griffin, M.J.: Handbook of human vibration. Academic press (2012)
7. Haddadin, S., Haddadin, S., Khoury, A., Rokahr, T., Parusel, S., Burgkart, R., Bicchi, A., Albu-Schäffer, A.: On making robots understand safety: Embedding injury knowledge into control. AISTATS 31(13) (Nov 2012)
8. ISO, I.: ISO 13482:2014: Robots and robotic devices – Safety requirements for personal care robots. International Organization for Standardization (2011)
9. Jacobs, M.: videantis ≫ Handy list of automotive ADAS acronyms
10. Keviczky, T., Falcone, P., Borrelli, F., Asgari, J., Hrovat, D.: Predictive control approach to autonomous vehicle steering. In: ACC. IEEE (2006)
11. Kim, J.S.: Recent advances in adaptive MPC. In: ICCAS (2010)
12. Kong, J., Pfeiffer, M., Schildbach, G., Borrelli, F.: Kinematic and dynamic vehicle models for autonomous driving control design. In: IV. IEEE (2015)
13. Lefevre, S., Carvalho, A., Borrelli, F.: A Learning-Based Framework for Velocity Control in Autonomous Driving. IEEE T-ASE 13(1) (Jan 2016)
14. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv:1509.02971 [cs, stat] (2015)
15. Lima, P.: Predictive control for autonomous driving. PhD Thesis, PhD thesis, KTH, 2016. Unpublished thesis (2016)
16. Maciejowski, J.M.: Predictive control: with constraints. Pearson education (2002)
17. Morari, M., H. Lee, J.: Model predictive control: past, present and future. Computers & Chemical Engineering 23(4–5), 667–682 (May 1999)
18. Pomerleau, D.A.: Efficient training of artificial neural networks for autonomous navigation. Neural Computation 3(1), 88–97 (1991)
19. Shia, V., Gao, Y., Vasudevan, R., Campbell, K., Lin, T., Borrelli, F., Bajcsy, R.: Semiautonomous Vehicular Control Using Driver Modeling. IEEE ITS (2014)
20. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction, vol. 1. MIT press Cambridge (1998)
21. VENTURER: VENTURER Trial 1: Planned handover. Tech. rep. (May 2017)
22. Watkins, C.J., Dayan, P.: Q-learning. Machine learning 8(3-4), 279–292 (1992)
23. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. In: Reinforcement Learning, pp. 5–32. Springer (1992)